AN ALGORITHM FOR THE THREE-INDEX ASSIGNMENT PROBLEM

Egon Balas
Carnegie Mellon University
Pittsburgh, PA

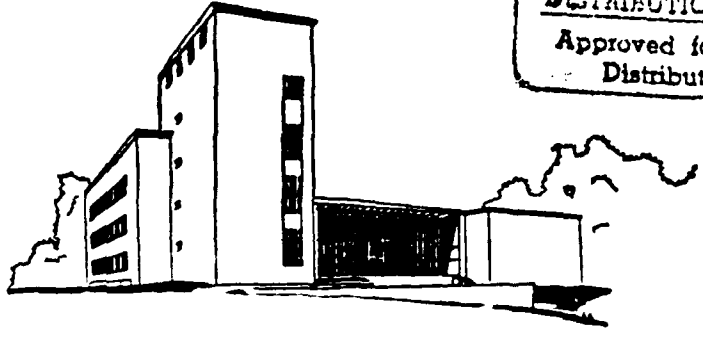
Matthew J. Saltzman
University of Arizona
Tucson, AZ

Carnegie Mellon University

PITTSBURGH, PENNSYLVANIA 15213

GRADUATE SCHOOL OF INDUSTRIAL ADMINISTRATION

WILLIAM LARIMER MELLON, FOUNDER



W.P. #88-89-23

Management Science Research Report No. 550

(1)

AN ALGORITHM FOR THE THREE-INDEX ASSIGNMENT PROBLEM

Egon Balas
Carnegie Mellon University
Pittsburgh, PA

Matthew J. Saltzman
University of Arizona
Tucson, AZ

September 1988

The research underlying this report was supported by Grant ECS-8601660 of the National Science Foundation, Contract N00014-85-K-0198 with the Office of Naval Research. Reproduction in whole or in part is permitted for any purpose of the U.S. Government.

Management Science Research Group
Graduate School of Industrial Administration
Carnegie Mellon University
Pittsburgh, PA 15213

Abstract

We describe a branch and bound algorithm for solving the axial three-index assignment problem. The main features of the algorithm include a Lagrangian relaxation incorporating a class of facet inequalities and solved by a modified subgradient procedure to find good lower bounds, a primal heuristic based on the principle of minimizing maximum regret plus a variable depth interchange phase for finding good upper bounds, and a novel branching strategy that exploits problem structure to fix several variables at each node and reduce the size of the total enumeration tree. Computational experience is reported on problems with up to 78 equations and 16,376 variables. The primal heuristics were tested on problems with up to 210 equations and 343,000 variables.

1. INTRODUCTION

In this paper we present an algorithm for the axial three-index assignment problem (AP3). AP3 can be stated as a 0-1 programming problem as follows:

$$\text{Min} \sum_{i\in I} \sum_{j\in J} \sum_{k\in K} c_{ijk} x_{ijk}$$

subject to

$$\sum_{j\in J} \sum_{k\in K} x_{ijk} = 1 \qquad \forall i \in I$$

$$\sum_{i\in I} \sum_{k\in K} x_{ijk} = 1 \qquad \forall j \in J$$

$$\sum_{i\in I} \sum_{j\in J} x_{ijk} = 1 \qquad \forall k \in K$$

$$x_{ijk} \in \{0,1\} \qquad \forall \{i,j,k\} \in I \times J \times K$$

where I, J and K are disjoint sets with $|I| = |J| = |K| = n$. We will sometimes use the notation $S = I \times J \times K$, and write $s \in S$ for $\{i,j,k\} \in I \times J \times K$.

AP3 is a close relative of the *(axial) 3-dimensional transportation problem* (TR3), in which the right-hand sides of the constraints can be any positive integers, the sets I, J, K are not necessarily equal in size, and the integrality constraints are relaxed. This is in turn a generalization of the well-known transportation problem, a special case of which is the simple assignment problem. This and other formulations of TR3 were first studied by Schell [15]. For references concerning these problems see [3].

Applications of AP3 mentioned in the literature include the following (Pierskalla [14,13]):

- In a rolling mill, schedule ingots through soaking pits (temperature stabilizing baths) so as to minimize idle-time for the rolling mill (the next stage in the process).

- Find a minimum cost schedule of a set of capital investments (*e.g.*, warehouses or plants) in different locations at different times.
- Assign military troops to locations over time to maximize a measure of capability.
- Launch a number of satellites in different directions at different altitudes to optimize coverage or minimize cost.

In [3], AP3 is shown to be equivalent to the problem of finding a minimum-weight exact clique cover in a complete tripartite graph.

AP3 is known to be NP-complete [8]. Obviously, AP3 is a special case of the *set partitioning problem* (SPP):

$$\text{Max } \{cx: \ Ax = e, \ x \in \{0,1\}^q\} \tag{1}$$

where A is a matrix of zeros and ones and e is a vector of ones. For properties of (SPP) see the survey [2].

Among the early algorithms and heuristics for this problem are those of Pierskalla [14,13] and Leue [11] (the latter is related to the Hungarian algorithm for the (two-index) assignment problem, and to an algorithm by Vlach [17] for the planar three-index assignment problem). A primal-dual algorithm is described by Hansen and Kaufman in [9]. The exact algorithms described in the literature share a branching strategy based on fixing single variables to zero or one. These algorithms do not use any results from polyhedral combinatorics; in fact there have been no polyhedral studies of this problem in the literature before [3]. Furthermore, aside from [14], no study of heuristics for this problem has been undertaken. The study presented here differs from the earlier papers in all of these respects.

The main features of the algorithm described are:

- Instead of the LP-relaxation, bounds are computed using a Lagrangian relaxation which incorporates facets of the AP3 polytope. This relaxation is solved by a modified subgradient optimization procedure.

- New, efficient primal heuristics are used to obtain successively improved approximate solutions.
- A new branching rule is applied which produces smaller search trees. Instead of completely specified primal solutions, the terminal nodes are two-dimensional assignment problems.

The first two features are modeled after the set covering algorithm of Balas and Ho [1].

In Section 2 we describe the overall flow of control in the algorithm. A study of primal heuristics is presented in Section 3. The maximum-regret heuristic is shown to produce better solutions than several others, and a variable-depth interchange heuristic brings further improvements. Sections 4 and 5 describe a Lagrangian relaxation of AP3 and a modified subgradient optimization algorithm for solving the relaxation. The relaxation incorporates one class of facet inequalities described in [3]. Section 6 describes a greedy-plus-interchange heuristic for generating lower bounds. In Section 7 we describe the branching strategy, *i.e.* the rules for generating the subproblems to be solved at each node of the enumeration tree, and for backtracking. Finally, Section 8 presents the results of computational experiments with an implementation of the algorithm. Our program is shown to compete effectively with the algorithm of Hansen and Kaufman, the only other procedure for which more than anecdotal experience is available.

2 OUTLINE OF THE ALGORITHM

The algorithm begins with the original problem on the list of unsolved subproblems. It proceeds to apply two primal heuristics and subgradient optimization on the Lagrangian dual. If it is unable to prove that the heuristic solution is optimal, the algorithm replaces the original problem

with two subproblems, each of which has fixed to zero one of two mutually exclusive, jointly exhaustive subsets of the variables in the support of a pair of rows of the constraint matrix. One of these is selected to be the next problem examined and the process is repeated (sometimes using a dual heuristic instead of subgradient optimization). If a subproblem can be shown to have no solution better than the best primal feasible solution found so far, that problem is discarded. All unsolved problems are then examined; if any can be discarded, they are deleted, and a "most promising" subproblem is selected to be examined next. When all outstanding subproblems have been deleted from the list, the current best solution is optimal.

The branching strategy is designed so that when enough blocks of variables are fixed, the remaining subproblem is a (two-dimensional) assignment problem, and can be solved to optimality in $O(n^3)$ time. These subproblems are the terminal nodes of the enumeration tree.

An outline of the algorithm in mock Pascal form follows. Text in braces {...} is commentary. Text in italics summarizes steps that are elaborated later or do not require further elaboration. The value of the best known primal solution is z_u. The best known lower bound at the current node is z_ℓ.

```
Algorithm AP3;
begin
  initialize;
  while unsolved nodes remain do {main loop}
    if current node not terminal then begin
      apply primal heuristics;
      update z_u and current solution;
      if  z_ℓ > z_u -1  then
```

```
            backtrack
         else begin
            apply subgradient or dual heuristics;
            update z_ℓ;
            if  z_ℓ > z_u - 1  then
               backtrack
            else
               create subproblems and branch
         end
      end
      else begin {current node is terminal}
         solve the terminal subproblem;
         update z_u and current solution;
         backtrack
      end
   end.
```

The algorithm as implemented is designed to solve problems with nonnegative integer costs. Some features of the algorithm rely on this assumption, particularly in the dual procedures. We will point out these dependencies as they arise. Note that any problem with integer costs can be "reduced" to a problem with nonnegative integer costs by adding a positive amount to each cost in the support of any row. This transformation does not affect the optimal solution. Initialization of the lower bound to zero is valid only for nonnegative costs. The bounding test $z_\ell > z_u - 1$ is dependent on the assumption of integer costs.

3 PRIMAL HEURISTICS

3.1 Constructing a Solution

Several primal heuristics were considered and tested for use in the primal phase of the algorithm. In this section we describe the heuristics and the results of computational tests.

We tested four heuristics. The first, *DIAGONAL*, is to simply set $x_{iii} = 1$ for $i = 1,\ldots,n$, and set all other $x_{ijk} = 0$. This is analogous to the Northwest Corner rule for starting the Transportation Simplex Algorithm (see, *e.g.*, [10]), and for randomly generated problems (such as those tested here) is equivalent to randomly selecting a solution.

The second rule tested was the *GREEDY* heuristic: The variable with smallest cost is selected, the three rows thus covered (and the variables in their respective supports) are deleted, and the process is repeated.

The third rule, *REDUCED-COST*, performs reductions on the costs, subtracting the largest cost in each row (*i.e.* the largest of the costs of variables in the support of each row) from all costs in that row. Then the element with the most negative reduced cost is selected, and the process is repeated. This rule is analogous to Russell's approximation method for starting the Transportation Simplex Algorithm (see, *e.g.*, [10]).

Finally, we have the *MAX-REGRET* heuristic: The difference between the two smallest costs associated with variables in the support of each row is calculated (the *regret* for that row, so called because it represents the minimum penalty for not choosing the smallest cost in the row). The element with smallest cost in the row with largest regret is selected, and the process is repeated. This method is analogous to Vogel's approximation for starting the Transportation Simplex Algorithm (see, *e.g.*, [10]).

The heuristics were tested on sets of randomly generated problems, twenty problems to a set, for n = 4, 6,...,20. Costs were generated from a uniform distribution of integers, $0 \le c_{ijk} \le 100$. The results are summarized in Table 1. The results demonstrate that the *MAX-REGRET* heuristic dominates the others, in terms of average solution quality (the columns labeled *Mean* in Table 1) and frequency with which it produced the best solution (the columns labeled *Wins*). In terms of computation time, the three "smart" rules were each implemented as $O(n^4)$ procedures. The *REDUCED-COST* rule took approximately 3.5 times longer than *GREEDY*, and the *MAX-REGRET* rule took approximately 8.75 times longer than *GREEDY* (*DIAGONAL* took an insignificant amount of time in all cases). The average time for *MAX-REGRET* with n = 4 was 10 milliseconds; with n = 20, *MAX-REGRET* averaged 3 seconds (implemented in **VAX C** on a **DEC VAX 11/780 under VAX/VMS**). *GREEDY* and *MAX-REGRET* can be implemented in $O(n^3 \log n)$ time by sorting the variables by cost beforehand, but the constant factor associated with sorting is comparatively high. We tested a sorted-cost version of *MAX-REGRET* using *QUICKSORT*, and this version finally overtook its $O(n^4)$ counterpart at around n = 19. Our algorithm does incorporate a sorting phase. The cost of sorting is not a problem in our case, first because the heuristic is to be

	DIAGONAL		GREEDY		RED-COST		MAX-REGRET	
n	Wins	Mean	Wins	Mean	Wins	Mean	Wins	Mean
4	0	210.65	7	80.45	10	69.40	12	68.80
6	0	276.50	2	90.05	6	75.00	14	55.15
8	0	391.75	4	92.15	3	90.75	13	58.45
10	0	504.20	3	78.50	3	81.10	14	60.15
12	0	646.45	3	90.85	4	83.00	13	51.95
14	0	663.25	4	89.20	2	83.55	14	50.05
16	0	795.15	2	85.30	1	73.95	17	42.00
18	0	892.60	3	79.00	2	76.10	15	45.35
20	0	1036.45	1	77.00	4	68.70	15	39.25

Table 1: Results of Comparative Tests of Primal Heuristics

applied several times to the same problem, so the sort need only be done once (each iteration of each heuristic is quite fast on sorted data); and second, because the heuristics are a relatively inexpensive part of the algorithm. The heuristics can be applied in a straightforward way to the subproblems in the enumeration tree, although it is possible that they will fail to find a feasible solution. To minimize this possibility *MAX-REGRET* was modified so that if only one element is left in a row at any time, that element is always selected. If the heuristic still fails to find a solution, the step is abandoned, and the value of the heuristic solution set to infinity.

3.2. Local Improvement

It is possible to apply local interchange heuristics to a feasible solution, which may succeed in finding a "nearby" solution with a better objective function value. For any pair of elements in a solution, $x_{i_0 j_0 k_0}$ and $x_{i_1 j_1 k_1}$ there are three possible ***pairwise interchanges***, replacing these two elements with elements with interchanged i-, j- or k-indices. If the sum of the costs of the two new elements is lower that the sum of the costs of the two original elements, then the exchange is carried out (if more than one interchange yields an improvement, the one that gives the maximum improvement is applied). The process can be applied repeatedly until no interchange takes place. This procedure may be applied after every successful application of the heuristic, after solving the assignment problem at each terminal node, and any time a primal-feasible solution is generated in the subgradient optimization procedure.

There are three extensions to the pairwise interchange heuristic that deserve mention. First, ***triple interchanges, etc.***, can be defined analogously to pairwise interchanges, though the computational burden becomes exponentially

heavier. Second, there is the concept of ***simulated annealing*** investigated by Burkard and Rendl [4] for the Quadratic Assignment Problem, and by Skiscim and Golden [16] and others for the Traveling Salesman Problem.

The algorithm we have implemented includes a third type of interchange heuristic, based on the Lin-Kernighan heuristic for the traveling salesman problem [12]. In the context of AP3, this ***variable-depth interchange*** heuristic works as follows: Start with a feasible solution, and set the ***total gain*** to G = 0. Select arbitrarily a variable that has the value 1 in the solution, say $x_{i_0j_0k_0}$. For each of the remaining variables $x_{ijk} = 1$, evaluate the possible interchanges of $x_{i_0j_0k_0}$, namely those formed by setting $x_{i_0j_0k_0} = x_{ijk} = 0$ and either $x_{i_0j_0k_0} = x_{ijk_0} = 1$, $x_{i_0j_0k_0} = x_{ij_0k} = 1$, or $x_{ij_0k_0} = x_{i_0jk} = 1$. Select the variable $x_{i_1j_1k_1}$ that maximizes the ***gain***, defined as $g((i_0,j_0,k_0), (i,j,k)) = c_{i_0j_0k_0} - \min\{c_{i_0j_0k}, c_{i_0jk_0}, c_{ij_0k_0}\}$. If the sum of the gain and total gain is positive, evaluate the cost of the solution with the interchange performed. If the new solution is an improvement over the current solution, then record the new one and continue. Set $x_{i_0j_0k_0} = 0$ and the variable for which the gain is maximized to 1 (suppose this variable is $x_{i_1j_0k_0}$). Set G to the sum of G and the gain. Repeat the process, setting $x_{i_1j_1k_1} = 0$ and considering interchanges between the other variable in the interchange ($x_{i_0j_1k_1}$, for example) and variables that have not been selected before. Continue as long as the sum of the gains is positive. If no improved solution is found starting with $x_{i_0j_0k_0}$, then try sequences starting with each of the other variables in the solution in turn. Repeat the entire process as long as improvements are found.

3.3 Computational Tests with Primal Heuristics

Table 2 shows the results of the initial application of the primal heuristics on the problems solved to optimality. The optimal solution (z^*) and the results of applying *GREEDY* (z^g), *MAX-REGRET* (z^h) and *MAX-REGRET* followed by *VARIABLE-DEPTH INTERCHANGE* (z^i) are displayed. The column labeled *int.* indicates the number of interchanges considered in all sequences. For each value of n, the average of each value over five problems solved to optimality is given (three problems for n = 26).

In our initial tests, the full algorithm spent roughly three times as long in the *VARIABLE-DEPTH INTERCHANGE* phase as in the *MAX-REGRET* phase.

n	z^*	z^g	z^h	z^i	int.
4	42.2	53.6	52.6	43.2	3.2
6	40.2	90.2	76.0	45.4	10.6
8	23.8	81.4	59.6	33.6	14.4
10	19.0	84.4	50.8	40.8	17.4
12	15.6	87.0	40.2	24.0	22.0
14	10.0	86.4	64.0	22.4	39.2
16	10.0	78.2	58.8	25.0	55.0
18	6.4	62.4	21.8	17.6	15.2
20	4.8	77.4	75.8	27.4	80.2
22	4.0	93.4	47.8	18.8	52.8
24	1.8	91.0	59.1	14.0	71.0
26	1.3	107.3	36.0	15.7	42.0

Table 2: Primal Heuristic Performance on Problems Solved to Optimality

4. THE LAGRANGIAN DUAL

A valid lower bound for a minimization problem can be found by solving to optimality a relaxation of the problem, with some of the constraints deleted. In general integer programming, frequently the integrality constraints are relaxed and the solution to the *LP-relaxation* is used as a bound. For problems with special structure, a useful alternative is to relax

some or all of the linear constraints by taking them into the objective function with Lagrange multipliers. The multipliers can then be given values such that the solution to the relaxation with the Lagrangian objective function gives the same bound as the LP-relaxation. The advantage of this *Lagrangian relaxation*, sometimes called *Lagrangian dual*, is that a good approximation to this bound can often be computed with much less effort than that involved in solving the LP. One technique for solving the Lagrangian relaxation is *subgradient optimization*.

4.1 A Lagrangian Relaxation of AP3

We considered four "natural" relaxations of AP3:

1. All of the original problem constraints (except for integrality constraints) are taken into the objective function. An optimal solution to the relaxation (for fixed values of the multipliers) is obtained by setting to one each variable for which the reduced cost is negative, and setting the remaining variables to zero.
2. All constraints (except integrality) are taken into the objective function and replaced by the single constraint that exactly n variables be set to one. An optimal solution to this relaxation is obtained by simply setting to one the n variables with smallest reduced cost, and setting all others to zero.
3. The two sets of equations corresponding to the ground sets J and K are taken into the objective function. An optimal solution to the relaxation is constructed by setting to one, for every $i \in I$, the variable in the support of row i with the smallest reduced cost.
4. The set of equations corresponding to the ground set I is taken into the objective function. An optimal solution to this relaxation is constructed

by solving an assignment problem over the ground sets J and K, with the cost for each (j,k)-pair given by the minimum reduced cost among all triplets containing the pair (j,k) (*i.e.* the minimum is taken over $i \epsilon I$).

In all of the above relaxations the Lagrange multipliers are unconstrained in sign, since they multiply equality constraints. As we pointed out above, the exact optimum of the Lagrangian dual is equal to the optimum for the LP relaxation in each case. In preliminary tests with the subgradient optimization algorithm, all the relaxations converged at comparable rates to a very close approximation of the LP optimum. Although the overall complexity of each of these steps can be shown to be $O(n^3)$ (except for relaxation 2, which requires $O(n^3 \log n)$), the constant factor is lowest for relaxation 3. Nevertheless, we chose to incorporate relaxation 4 in the algorithm for reasons we will explain below. Our Lagrangian dual is

$$\underset{u}{\text{Max}}\ L(u) \qquad (2)$$

where

$$L(u) = \underset{x}{\text{Min}} \sum_{i \epsilon I} \sum_{j \epsilon J} \sum_{k \epsilon K} (c_{ijk} - u_i) x_{ijk} + \sum_{i \epsilon I} u_i$$

subject to

$$\sum_{i \epsilon I} \sum_{k \epsilon K} x_{ijk} = 1 \qquad \forall j \in J$$

$$\sum_{i \epsilon I} \sum_{j \epsilon J} x_{ijk} = 1 \qquad \forall k \in K$$

$$x_{ijk} \in \{0,1\} \qquad \forall i,j,k$$

The costs for the assignment problem over J and K are computed as $c'_{jk} = \min_{i \epsilon I} (c_{ijk} - u_i)$.

4.2 Incorporating Facet Inequalities

The lower bound on the value of an integer program can often be considerably strengthened by adding to the constriant set valid inequalities that cut off solutions to the LP-relaxation. The strongest such inequalities are the *facets* of the convex hull of integer solutions to the program. Although for *NP*-hard problems such as AP3, the number of such facets may be very large, they need not be added all at once; if the optimal LP solution is known, it may be possible to detect a violated facet in a reasonable amount of time. The violated facet can be added to the constraint set, and the LP re-optimized. We have described several classes of facets for the three-index assignment polytope [3]. In the algorithm presented here, we incorporate one such class in the relaxation, namely the class of inequalities derived from cliques of *type 2*, as defined in [3], of the intersection graph associated with AP3. These cliques are defined as

$$Q_2(i^*,j^*,k^*) =$$

$$\{(i,j,k):\ i = i^*,\ j = j^* \text{ or } i = i^*,\ k = k^* \text{ or } j = j^*,\ k = k^*\}.$$

The facet inequality is

$$\sum_{s \in Q_2(i^*,j^*,k^*)} x_s \leq 1.$$

These constraints can also be taken into the objective function with Lagrange multipliers, but since there are n^3 of them, the computational burden of computing the reduced costs in an instance of the relaxation is heavy. Again, only a subset of the inequalities is likely to be active (*i.e.* to have nonzero dual multiplier value) at any point. We need to develop a method for detecting the inequalities that are likely to be active and including them. The Lagrangian function becomes:

$$\sum_{i\in I}\sum_{j\in J}\sum_{k\in K}(c_{ijk}-u_i)x_{ijk} + \sum_{i\in I}u_i - \sum_{s\in T}\rho_s\left(\sum_{t\in Q_2(s)}x_t - 1\right) \qquad (3)$$

where $\rho_s \geq 0$, and $T \subseteq S$ is the subset of the variables that generate active clique constraints.

Which constraints should be chosen? Solving the LP-relaxation of AP3 would give us a basic solution having at most 3n - 2 non-zero variables, and if this solution were fractional, we could look for inequalities violated by it. In the subgradient technique, we do not have a primal feasible LP solution at hand, so we cannot identify violated inequalities directly. Instead we consider the solution to the relaxation corresponding to the best bound generated by the subgradient procedure. Such a solution satisfies exactly the constraints corresponding to the ground sets J and K, but may under- or over-cover constraints corresponding to ground set I. Any pair of variables equal to one in the solution which, taken together, over-cover a constraint in ground set I, also violate exactly two type-2 facet inequalities. Suppose two such variables are $x_{i_0j_0k_0} = x_{i_0j_1k_1} = 1$. Then x violates the facet inequalities corresponding to $Q_2(i_0,j_0,k_1)$ and $Q_2(i_0,j_1,k_0)$. These cuts are incorporated into the objective function, and the subgradient procedure then continues as described below.

5. THE SUBGRADIENT OPTIMIZATION PROCEDURE

We use subgradient optimization to solve the Lagrangian dual problem, which we restate as

$$\max_{\pi} L(\pi)$$

with

$$L(\pi) = \min_{x\in F} c(\pi)x + k(\pi).$$

Here F is the set of feasible solutions to the constraint set of (2), π is the vector of dual variables, $k(\pi)$ is a constant with respect to x for a given π, and $c(\pi)$ is the Langrangian objective coefficient vector of x. In the case of our original relaxation of AP3, $\pi = u$. When facet inequalities are added, $\pi = (u,\rho)$ and $\rho \geq 0$ is required. We begin with $\pi^0 = 0$ and iterate the following steps (m is the iteration counter):

1. Solve $\min_{x \in F} c(\pi^m)x$ to determine $L(\pi^m)$ and a subgradient direction μ^m.
2. Determine a direction s^m based on μ^m, and a step length t_m.
3. Set $\pi^{m+1} = \pi^m + t_m s^m$, $m = m + 1$.

The procedure continues until some stopping criteria (described below) are met. We have described in section 4.1 the procedure for solving the relaxation. The remaining steps are discussed below. Finally we describe how the procedure interacts with the branching process.

5.1 Choosing a Direction

The subgradient for the original relaxation is μ, where $\mu_i = 1 - \sum_{j \in J} \sum_{k \in K} x^*_{ijk}$ for $i \in I$ and x^* is the optimal solution to the relaxation. When the relaxation includes facet inequalities, $\mu = (\mu^u, \mu^\rho)$ where $\mu^\rho_q = \max\{0, \sum_{(i,j,k) \in Q_2(q)} x_{ijk} - 1\}$. This is one candidate for the direction (*i.e.* $s^m = \mu^m$). Another possibility, suggested by Camerini, Fratta and Maffioli [5], is a modified subgradient that takes into account the recent history of directions to reduce the tendency of the subgradient to "zig-zag", alternating between two directions without making significant progress toward the dual optimum. The direction [5] is defined componentwise as $s^m_p = \mu^m_p + \beta_m s^{m-1}_p$ where μ^m is the subgradient, and β_m is a scalar weight on the direction from the previous iteration, defined as

$$\beta_m = \begin{cases} -\theta \dfrac{s^{m-1}\mu^m}{\|s^{m-1}\|^2} & \text{if } s^{m-1}\mu^m < 0, \\ 0 & \text{otherwise.} \end{cases}$$

The parameter θ controls the angle between s^{m-1} and s^m, and must be between zero and two. The value suggested in [5] is $\theta = 1.5$, which we used here. The idea behind the choice of s^m is that β_m is greater than zero when the actual gradient direction μ^m and the direction in the preceding step s^{m-1} form an obtuse angle, favoring the "persistent components" of the subgradient over its "alternating components".

Our early experiments confirm that, for our problem, this *smoothed subgradient* direction is indeed an improvement over the subgradient alone. At some nodes of the enumeration tree, the dual procedure using the subgradient direction was unable to improve the bound given by the penalty function (described below), within a fixed number of iterations. The modified direction was often able to find an improvement in these cases within the same number of iterations.

5.2 Computing a Step Length

The step length is defined as

$$t_m = \lambda \frac{z^u - z_m}{\|s^m\|^2}$$

where z^u is the current upper bound and z_m is the value of the current solution to the relaxation. In order to guarantee convergence (*i.e.* $t_m \to 0$ as $m \to \infty$), the step length contains a multiplicative parameter $0 \le \lambda \le 1$ (see [5] for justification) that shrinks over time. The rate at which λ shrinks is an important factor in determining the performance of the subgradient optimization procedure. Too slow a rate may cause the procedure to run on

without improving the bound. Too rapid a rate may cause termination of the procedure before the optimum is achieved. We use the following rule: Start with $\lambda = \lambda_0$ where

$$\lambda_0 = \begin{cases} 0.5 & \text{if } 0.95 \le z_\ell/z_u \\ 0.75 & \text{if } 0.90 \le z_\ell/z_u < 0.95 \\ 1.0 & \text{if } z_\ell/z_u < 0.90 \end{cases} .$$

Halve λ after some number of iterations since the last improvement in the bound. The number of iterations is set to n at the start. After an improvement of 1%, an additional n/2 iterations are allowed for additional improvement. The idea behind this rule is that, if improvements are being made in the bound, we want to continue to take longer steps and not force early termination. After some time with no improvement, the step length should be made smaller. Of course these rules are heuristic; although important, the influence of rules controlling λ is not well understood.

5.3 Stopping

The subgradient procedure is terminated if too many iterations occur without an acceptable improvement. Specifically, we allow 2n iterations from the start of the procedure to gain an improvement of at least 5%; subsequently n/2 iterations are allowed after each 5% improvement. The procedure is also terminated if a solution to the relaxation is found to be primal feasible or (as a last resort) if the step length becomes too small ($t_m < 10^{-8}$). If a primal feasible solution is detected, it is checked for optimality for the subproblem, by testing complementary slackness conditions for the currently active cuts. If the solution is optimal, the current node can be discarded. In any case, the solution is improved with sequential interchanges and the upper and lower bounds are updated.

5.4 Generating Primal Feasible Solutions

Given a solution to the relaxation for some fixed π, a primal feasible solution to AP3 can be generated in the same fashion as is done at a terminal node in the enumeration tree. The variables set to one in the solution may be taken to specify an assignment of the elements of J to the elements of K, J and K being the two ground sets for which the constraints are satisfied. Once this assignment has been specified, the optimal allocation of elements of I to (j,k)-pairs is a (2-index) assignment problem.

After each call to the subgradient procedure, the relaxation solution associated with the best lower bound is made feasible in this manner, and the primal solution constructed is improved by applying to it *VARIABLE-DEPTH INTERCHANGE*. If it is an improvement over the incumbent solution, the incumbent is replaced.

5.5 Incorporating Facets

For the first application of the subgradient routine, all dual variables are set to zero, and the direction vector is set to zero. Initially, no cuts are active. After the first execution of the procedure, any type-2 clique facets violated by the solution to the relaxation corresponding to the best lower bound generated are added to the active list, with the associated multiplier set to zero. The subgradient procedure is then applied again, with the prior ending solution and direction used as the starting point.

After each subsequent execution of the subgradient procedure, the list of active cuts is checked against the new solution to the relaxation. If any cuts in the list are no longer active (*i.e.* have zero weight), they are dropped from the list. If any of the dropped cuts were previously active (had

positive weight), then the cut list is updated (*i.e.* any cuts violated by the best-bound solution are added to the list), and the subgradient procedure is applied again. The strategy of not updating the active cut list unless a cut is dropped represents a compromise between the computational effort involved in updating the active cut list ($O(n^2 \times$ number of cuts in the list)), and the desirability of keeping the list as up to date as possible. The subgradient procedure is then repeated. If no formerly active cuts are dropped, the algorithm continues.

The performance of the subgradient optimization procedure with and without the cuts is illustrated in Table 3 on the problem set described earlier (at the root node of the search tree). Here z^* is the value of the optimal solution, z_ℓ^o and z_ℓ^c are the lower bounds obtained by the procedure without cuts and with cuts, respectively, and the "% gap" is $((z_\ell^c - z_\ell^o)/(z^* - z_\ell^o)) \times 100$.

n	z^*	z_ℓ^o	z_ℓ^c	% gap	# cuts
4	42.2	39.96	40.77	26.96	1.6
6	40.2	35.25	35.75	39.12	5.2
8	23.8	18.55	19.09	40.67	5.6
10	19.0	15.58	16.43	43.01	8.4
12	15.6	13.63	13.78	52.19	15.2
14	10.0	6.67	7.20	41.98	17.6
16	10.0	6.50	6.67	36.21	15.6
18	6.4	3.59	3.78	35.46	15.2
20	4.8	1.49	1.88	28.37	25.6
22	4.0	1.45	1.70	30.32	22.0
24	1.8	0.15	0.23	17.47	22.4
26	1.3	0.11	0.19	9.60	51.3

Table 3: Performance of Subgradient Optimization

6. DUAL HEURISTICS

Because the subgradient procedure is expensive, we chose to alternate it with some dual heuristics, according to criteria described in Subsection 7.4 below. In this section we describe the greedy and interchange heuristics used as alternatives to the subgradient procedure.

The dual of the LP relaxation of AP3 is

$$\text{Max} \sum_{i \in I} u_i + \sum_{j \in J} v_j + \sum_{k \in K} w_k$$

subject to

$$u_i + v_j + w_k \leq c_{ijk} \qquad \forall i \in I,\ j \in J,\ k \in K$$

The first phase of the procedure is a simple greedy heuristic. Let $\bar{c}_{ijk} = c_{ijk} - u_i - v_j - w_k$. Then as long as there are free variables, find the triplet (i,j,k) for which $\bar{c}_{ijk}$ is minimized and allocate to each free variable in the (dual) row an equal fraction of $\bar{c}_{ijk}$. This procedure is applied at the root of the enumeration tree, and at the interior nodes of the tree as described below.

The solution constructed by the greedy heuristic is improved using a procedure similar to that proposed by Fisher and Kedia [6] for set partitioning, but modified to take advantage of the special structure of AP3. For each dual variable u_{i^*}, $i^* \in I$, the variables associated with the other ground sets (J and K) are scanned to locate a maximal set of variables v_j and w_k with the property that (i) none of the selected variables appear together in a tight constraint; and (ii) none of the selected variables appear in a tight constraint together with any u_i, $i \neq i^*$. The dual solution can then be modified by decreasing the value of u_{i^*} and increasing the values of each of the selected variables v_j and w_k by equal amounts until some new constraint becomes tight. If the number of variables to be increased is m and the amount of increase is δ then the value of the dual solution is increased by $(m - 1)\delta$.

7. BRANCHING AND SUBPROBLEMS

In this section we describe the rules for branching (defining the constraints imposed and the subproblems generated when a branch is taken). We also discuss the criteria for choosing a particular branch at a given node, and the backtracking strategy.

7.1. Subproblem Definition

The number of feasible solutions to an order-n axial AP3 problem is easily seen to be $(n!)^2$. The common branching strategies for general integer programs, and all the branching strategies used in the published algorithms for AP3, are designed so that a terminal node in the enumeration tree represents a complete solution to the original problem. The number of terminal nodes in the complete tree is thus $(n!)^2$. In addition, branching rules that specify setting a single variable to one typically require that several subproblems be generated from the current subproblem when branching takes place. The branching rules described in [14] and [11] produce $(n-i+1)^2$ subproblems at level i in the tree, each of which is an AP3 problem of order n - i. The maximum depth of such a tree is n - 1. The binary branching scheme, in which a single variable is set to one in one subproblem and zero in the other, produces a tree in which the depth ranges from n on a path where all subproblems arise from setting variables to one, to $n^3 - n$ on a path where they arise from setting variables to zero.

For general 0-1 programming, branching on *multiple choice constraints* of the form $\sum_{j \in Q} x_j = 1$ for a subset Q of the variables, is often done by splitting the set Q in half and fixing the variables in each half to zero in the respective branch. In the case of AP3, this strategy leads to an enumeration

tree of depth O(n log n) with the same $(n!)^2$ terminal nodes. The rule is attractive because it keeps the tree depth relatively small by locking several variables in each branch, but it does not exploit problem structure in any way.

The rule chosen for this algorithm also fixes several variables in each branch, but does not split the support of a row evenly. Instead it exploits problem structure in a way that complements the Lagrangian relaxation used in the dual procedures and reduces the number of terminal nodes in the tree. The depth of a path in the tree ranges from n - 1 to n^2 - n depending on whether each branch fixes the larger or smaller block of variables.

Let M_r denote the support of row $r \in R = I \cup J \cup K$. Consider the ground sets J and K. Any pair of indices $j_0 \in J$ and $k_0 \in K$ specifies a ***block*** of n variables, $M(j_0,k_0) = \{x_{ij_0k_0} : i \in I\}$. In fact the row corresponding to j_0 consists of n such sets, one for each possible k_0. Let P be a permutation of {1,...,n} and let $[j]_P$ denote the element in position j of P. If the variables in $\bar{M}(j,k) := (M_j \cup M_k) \setminus M(j,k)$ are fixed to zero for j = 1,...,n, $k = [j]_P$, the problem that remains when the columns corresponding to these variables are deleted is a (2-index) assignment problem, in which the elements of I are to be matched to (j,k)-pairs. There are n! possible assignment problems (corresponding to the n! permutations of J), so our search tree, when complete, has n! rather than $(n!)^2$ leaves. Each of these problems can be quickly solved to optimality.

Our branching rule is to choose a (j,k)-pair, and set the variables in M(j,k) to zero in one branch (the *weak side branch*), and those in $\bar{M}(j,k)$ to zero in the other (the ***strong side branch***). For any j_0, k_0, when the weak side constraints corresponding to $M(j_0,k)$ have been imposed for all $k \neq k_0$, the strong side constraint corresponding to $\bar{M}(j_0,k_0)$ is imposed immediately,

since imposing the weak side constraint for $M(j_0,k_0)$ would render the subproblem infeasible. When n strong side constraints have been imposed, a terminal node is reached, and the corresponding (2-index) assignment problem is solved by the shortest augmenting path method. (In our tests, no terminal nodes were ever generated for any of the problems solved to optimality.)

7.2 Branch Selection

How should the next index pair for branching be selected? We use a two level selection process. At the first level, we prefer to select from the set of index pairs with the property that in both subproblems the lower bound can be increased over that of the parent. The technique for finding this set, and a penalty that can be calculated for each subproblem is described below. Within this set, or if this set is empty, or if the penalty calculation cannot be performed, we select the index pair identifying the block for which the minimum cost of a free variable in the block is maximum. Note that variables can be fixed to zero before branching at each node if their reduced costs exceed the gap between the upper bound and the lower bound of the current node. In case of ties, the block with the largest number of free variables is chosen. The two subproblems corresponding to the selected (j,k)-index pair are created and the weak side problem is selected next. This "depth-first" strategy is pursued until a node on the path is discarded. Then a new problem is selected from the list of unsolved problems, in a manner described below. A child node that can be discarded based on a penalty is simply not created.

7.3 Backtracking and Subproblem Selection

If the current node is discarded or the weak subproblem is not created, either because the branch is infeasible or because it is discarded based on

penalties, then a new subproblem must be selected from the remaining unsolved problems. Our algorithm uses a ***flexible backtracking strategy***, in which the list of subproblems is scanned for the "most promising" subproblem, which is selected next. As a measure of promise for a subproblem, instead of simply using the lower bound, we use an adaptation of the ***best projection*** rule described by Forrest, Hirst and Tomlin [7] for general integer programming. The measure of promise is a projection from the lower bound of the current subproblem to the lower bound at a terminal subproblem, defined as:

$$z_p = z_\ell + d(z_u - z_0)/d_0 \qquad (3)$$

where z_u is the current upper bound, z_0 and z_ℓ are the lower bounds at the root of the search tree and at the current node, respectively, d_0 is a measure of the distance from the root of the search tree to a terminal node, and d is the same measure for the current node. To measure the distance to a terminal node, we use the number of weak side branching constraints needed in order to transform the current subproblem into a completely specified assignment problem. For the original problem (at the root of the enumeration tree) this number is $d_0 = n^2 - n$. In general, the number is $d = n^2 - n - w$, where w is the number of weak side constraints required to form the current subproblem from the original problem. (Each arc on the path from the root to the current node signifies the imposition of a strong or weak side constraint. Each strong side constraint may be thought of as imposing 2(n-1) weak side constraints, some of which may duplicate constraints imposed at earlier levels in the tree. These duplications are not counted in the distance calculation.)

The backtracking routine thus scans the entire list of unsolved subproblems. If any node can be discarded (because the upper bound has been updated since the subproblem was created), that subproblem is deleted from the list. If the subproblem is still active, the projection of its lower bound is

computed, and the problem with the best projection is made the new current subproblem.

7.4 Applying Subgradient and Dual Heuristics at Subproblems

The bounds produced by the subgradient procedure are significantly more accurate than the greedy bounds, but the improvement comes at a substantial cost in terms of computational effort. As a compromise, we apply the following strategy: At the first node in the branch and bound tree, both the greedy-plus-interchange and subgradient procedures are applied, and the best bound is used as the lower bound. At subsequent nodes either the dual greedy or the subgradient procedure is applied, depending on the node's position in the tree:

- If the node was generated in a strong side branch, then the subgradient procedure is applied, using the solution of the nearest ancestor at which the subgradient procedure was previously applied as the starting solution. The same rule is applied after 2n - 1 weak side branches, since 2n - 1 weak side branches fix the same number of variables as a single strong side branch.
- At other nodes, the greedy-plus-interchange heuristic is applied, based on the original costs. (We also tried using the reduced costs including the cuts and associated weights from the nearest ancestor at which the subgradient procedure was applied, but this did not seem to improve the overall performance of the algorithm.)

When branching, the best dual solution found for an ancestor of the current subproblem is adopted as the starting solution to the subgradient optimization procedure for the subproblem, and the initial direction vector is set to the last smoothed subgradient of the ancestor.

In order to improve the lower bound on a child problem, and to determine if the subgradient optimization procedure might lead to an improved bound for the child, a penalty function is computed when the branch index is selected. The penalty is defined as the difference between the lower bound of the parent node and the solution to the Lagrangian relaxation of the child problem using the initial dual variable values. The penalty is clearly zero if none of the variables in the best relaxation solution to the parent problem are excluded in the child as a result of variable-fixing in the branching phase. In this case, if the subgradient procedure has indeed converged at the parent node then it is unlikely that any improvement can be obtained by continuing the procedure on the child. Thus the dual procedure is not applied to the child if the penalty is zero. If any variable set to one in the parent's relaxation solution is fixed to zero by the branching procedure, then the penalty is calculated as described above and the child's lower bound is set to the sum of the parent's lower bound and the penalty. If the child cannot be discarded on the basis of the penalty, it is placed on the unsolved problem list and the algorithm continues. We note that the penalty calculation only takes into account the variables fixed in $M(j,k)$ or $\bar{M}(j,k)$, and not variables that may be fixed by implication. It is thus possible that a variable actually is excluded in the child problem but not considered in the penalty calculation.

8 COMPUTATIONAL EXPERIENCE

8.1 The Algorithm as a Whole

We have implemented the algorithm on a DEC VAX 8650 in VAX C (under VAX/VMS). A set of random test problems was generated with the following characteristics: five problems for each even value of n ranging from 4 to 24

were generated with integer costs from a uniform distribution between 0 and 100. In addition, three with n = 26 with the same range of costs were solved (the remaining two problems in this set could not be solved in 30 minutes of CPU time each). The results of the tests are summarized in Table 4. The column *Secs* displays the total CPU seconds required to solve each problem. *Nodes* is the total number of nodes visited (*i.e.* subproblems solved by the heuristic), and *SGD* is the total number of subproblems to which the subgradient procedure was applied. *Iter* is the total number of subgradient iterations. *Depth* is the maximum depth of the branch and bound tree. The columns *% primal*, *% sgd* and *% dual* represent the percentages of CPU time spent in the primal heuristics, subgradient optimization and dual heuristics, respectively. Most of the time in the subgradient procedure is spent forming and solving the relaxations. In the primal heuristics, most of the time is spent in the sequential interchange procedure. Each line in the table represents the averages over the five problems in each set (three problems for n = 26).

n	n^3	Secs	% primal	% sgd	% dual	Nodes	SGD	Iter	Depth
4	64	0.03	12.86	48.58	5.72	2.4	1.60	9.00	1.60
6	216	0.16	12.66	54.80	9.52	8.2	4.20	38.80	5.00
8	512	0.84	8.20	74.90	7.44	18.6	9.20	111.20	6.20
10	1000	1.36	9.02	68.74	9.46	21.6	8.80	124.60	10.40
12	1728	2.19	6.32	72.80	8.08	22.6	10.60	123.40	12.40
14	2744	11.95	7.78	65.80	15.32	108.8	50.60	687.40	24.40
16	4096	39.90	8.34	63.12	20.92	270.6	98.20	1665.80	38.60
18	5832	55.30	7.04	66.56	19.80	277.4	85.60	1830.80	34.00
20	8000	169.29	5.32	67.80	20.20	625.2	223.40	4412.20	49.40
22	10648	371.52	4.98	68.66	21.18	1053.8	284.80	8432.00	73.40
24	13824	514.52	4.46	70.12	20.54	1177.8	466.00	10512.80	70.40
26	17576	624.00	4.27	63.87	28.00	1290.0	381.67	11456.33	98.00

Table 4: Computational Results

These results may be compared with the results of Hansen and Kaufman [9], reproduced in Table 5. It is difficult to compare actual computation times on their CDC 6400 and our VAX 8650, and we do not have access to the problems

they solved, although our problems are generated from the same distribution. They report times that increase by a factor of about six for each increase of n by two (over the range n = 4,...,12). Over the range n = 4,...24, our algorithm shows an increase by a factor of about three.[1] Also, the number of nodes evaluated approximately doubles for each increase of n by two.

n	# problems	CPU secs
4	5	0.59
6	5	2.75
8	5	10.59
10	5	60.27
12	5	359.77
16	1	674.69

Table 5: Computational Results from Hansen and Kaufman

There are some other observations that we can make regarding the performance of our algorithm. In particular, as the number of variables ($q = n^3$) increases, the ratio of the number of nodes of the search tree visited to the number of variables increases slowly from about 4% to about 10% over the range n = 4,...,24. Also, there are no instances in which the algorithm was required to solve a terminal assignment problem. This indicates that the bounding procedure is reasonably effective, and that the heuristics are finding the optimal solution, although not necessarily at the first application.

There is an observation that we made during the course of our preliminary experiments that is not reflected in the results shown here, but that

---

[1] Since only one example of a problem larger than n = 12 is provided in [9], it is impossible to gauge the performance of that algorithm on larger problems.

influenced design decisions during the development of the algorithm. That observation is that the primal and dual procedures that gave the best bounds at the root of the enumeration tree did not necessarily perform the best down in the subtrees. In particular, using these procedures often resulted in enumeration trees with more nodes than the procedures we finally selected, even though they gave better bounds at the start. Since the procedures that gave better bounds often required substantially more running time per application as well, the savings from not using them is increased still more.

The most expensive component of the algorithm is the relaxation solver in the subgradient procedure. As we pointed out in Section 5, the factors relating to the "optimal" number of subgradient iterations are not well understood. In addition, the behavior of the method when used in a branch and bound context, where it is started with a near-optimal solution to each subproblem, has not been extensively investigated. As in most studies to date, we have used *ad hoc* rules to determine the number of subgradient iterations. This is an issue which merits further careful experimental research, in the context of other classes of problems. Finally, some of the new aspects of this algorithm (especially the way facets are handled in the Langrangian dual procedure) can be employed in algorithms for other problems.

8.2 The Primal Heuristic as a Stand-Alone

We applied the *MAX-REGRET* and *VARIABLE-DEPTH INTERCHANGE* heuristics to a collection of larger problems, ranging from n = 20 to n = 70 (up to 210 equations and 343,000 variables). In view of the larger number of variables, these test problems were generated with costs ranging from 0-1000. Table 6 shows the results of these tests (averages over five problems for each value of n). The column labeled z^g is the result of the *GREEDY*

heuristic. The result of the *MAX-REGRET* heuristic is in the column labeled z^h. The *VARIABLE-DEPTH INTERCHANGE* procedure was applied to the solution produced by *MAX-REGRET*, and the results are given in the z^i column. The number of interchanges is reported in the int. column. The times reported (sec.) are CPU seconds for *MAX-REGRET* and *V.-D. INTERCHANGE* together, this time on a VAX 8600 running VMS. For problems of this size, the configuration of system memory has a significant impact on performance, *i.e.*, paging gets to be expensive, especially on a system not carefully tuned for memory-intensive applications. The lower bounds (z_ℓ) are the result of applying the subgradient procedure described below.

Our conclusion that *MAX-REGRET* is superior to *GREEDY* continues to be supported. Overall, *MAX-REGRET* gives values about 1/2 those of *GREEDY*. *VARIABLE-DEPTH INTERCHANGE* brings the solutions down again by a factor of roughly two. Over the 55 problems in the sample, *GREEDY* found a better solution than *MAX-REGRET* only seven times. In only ten problems was *VARIABLE-DEPTH INTERCHANGE* unable to improve on the solution found by *MAX-REGRET*. In these problems, the average number of interchanges was 24.4.

n	n^3	z^g	z^h	z^i	z_ℓ	int.	sec.
20	8000	823.0	569.6	266.2	98.94	75.6	0.16
25	15625	871.8	438.4	205.4	75.30	50.6	0.38
30	27000	918.4	416.2	166.0	63.98	99.6	0.82
35	42875	846.0	437.0	195.0	44.80	86.6	1.44
40	64000	1148.0	618.0	175.2	40.60	111.4	2.45
45	91125	988.0	420.2	186.8	34.00	105.6	4.18
50	125000	910.6	369.0	229.0	24.38	156.6	5.88
55	166375	854.0	356.8	183.2	15.18	187.4	9.16
60	216000	1107.0	403.4	137.4	10.08	108.6	12.69
65	274625	1064.8	354.8	132.0	5.90	87.6	26.57
70	343000	865.8	477.2	167.6	3.88	301.0	64.91

Table 6: Primal heuristic performance on large problems

We conclude that the combined *MAX-REGRET* plus *VARIABLE-DEPTH INTERCHANGE* heuristic is by itself a powerful practical approximation method for efficiently solving very large problems.

REFERENCES

[1] E. Balas and A. Ho, "Set Covering Algorithms Using Cutting Planes, Heuristics and Subgradient Optimization: A Computational Study." *Mathematical Programming*, 12:37-60, 1980.

[2] E. Balas and M. Padberg, "Set Partitioning: A Survey." *SIAM Review*, 18:710-760, 1976.

[3] E. Balas and M. J. Saltzman, "Facets of the Three-Index Assignment Polytope." Management Science Research Report 529, Carnegie Mellon University, October 1986.

[4] R. Burkard and F. Rendl, "A Thermodynam cally Motivated Simulation Procedure for Combinatorial Optimization Problems." Bericht 83-12, Institut fur Mathematik, Technische Universitat Graz, Graz, Austria, 1983.

[5] P. Camerini, L. Fratta, and F. Maffioli, "On Improving Relaxation Methods by Modified Gradient Techniques." *Mathematical Programming Study*, 3:26-34, 1975.

[6] M. Fisher and P. Kedia, "A Dual Algorithm for Large Scale Set Partitioning." Technical Report, The Wharton School, University of Pennsylvania, Philadelphia, PA, June 1986.

[7] J. Forrest, J. Hirst, and J. Tomlin, Practical Solution of Large Mixed Integer Programming Problems with UMPIRE." *Management Science*, 20:736-773, 1974.

[8] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, San Francisco, CA, 1979.

[9] P. Hansen and L. Kaufman, "A Primal-Dual Algorithm for the Three-Dimensional Assignment Problem." *Cahiers de CERO*, 15:327-336, 1973.

[10] F. Hillier and J. Lieberman, *Introduction to Operations Research*. Holden-Day, Inc., San Francisco, CA, 1980.

[11] O. Leue, "Methoden zur Lösung dreidimensionaler Zuordnungsprobleme." *Angewandte Informatik*, 154-162, April 1972.

[12] S. Lin and B. W. Kernighan, "An Effective Heuristic Algorithm for the Traveling Salesman Problem." *Operations Research*, 21:498-516, 1973.

[13] W. P. Pierskalla, "The Multidimensional Assignment Problem. *Operations Research*, 16:422-431, 1968.

[14] W. P. Pierskalla, "The Tri-Substitution Method for the Three-Dimensional Assignment Problem." *CORS Journal*, 5:71-81, 1967.

[15] E. D. Schell, "Distribution of a Product by Several Properties." In *Second Symposium in Linear Programming*, pages 615-642, DCS/Comptroller HQ. U.S. Air Force, Washington, DC, 1955.

[16] C. Skiscim and B. Golden, "Optimization by Simulated Annealing: A Preliminary Computational Study for the TSP." In *N.I.H.E. Summer School on Combinatorial Optimization*, Dublin, Ireland, 1983.

[17] M. Vlach, "Branch and Bound Method for the Three-Index Assignment Problem." *Ekonomicko-Matematicky Obzor (Czechoslovakia)*, 3:181-191, 1967.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ADA205172

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
MSRR-550		

4. TITLE (and Subtitle)
An Algorithm for the Three-Index Assignment Problem

5. TYPE OF REPORT & PERIOD COVERED
Technical paper, Sept 1988

6. PERFORMING ORG. REPORT NUMBER

7. AUTHOR(s)
Egon Balas
Matthew J. Saltzman

8. CONTRACT OR GRANT NUMBER(s)
N00014-85-K-0198

9. PERFORMING ORGANIZATION NAME AND ADDRESS
Graduate School of Industrial Administration
Carnegie Mellon University
Pittsburgh, PA 15213

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS

11. CONTROLLING OFFICE NAME AND ADDRESS
Personnel and Training Research Programs
Office of Naval Research (Code 434)
Arlington, VA 22217

12. REPORT DATE
September 1988

13. NUMBER OF PAGES
31

14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)

15. SECURITY CLASS. (of this report)

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)
3-index assignment
3-dimensional matching
subgradient optimization
max-regret heuristic
variable-depth interchange

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

We describe a branch and bound algorithm for solving the axial three-index assignment problem. The main features of the algorithm include a Lagrangian relaxation incorporating a class of facet inequalities and solved by a modified subgradient procedure to find good lower bounds, a primal heuristic based on the principle of minimizing maximum regret plus an interchange phase for finding good upper bounds, and a novel branching strategy that exploits problem structure to fix several variables at each node and reduce the size of the total enumeration tree. Computational experience is reported.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)